CMPT 432 Winter 2006/2007

Department of Computer Science
University of Saskatchewan

March 2, 2007
Number of Pages: 6
Time: 50 minutes
Total: 50 marks.

CAUTION - Candidates suspected of any of the following, or similar, dishonest practices shall be dismissed from the examination and shall be liable to disciplinary action

1. Having at the place of writing any communication devices, any books, papers or memoranda, calculators, audio or visual cassette players, or other memory aid devices.
2. Speaking or communicating with other candidates.
3. Purposely exposing written papers to the view of other candidates. The plea of accident or forgetfulness shall not be received.

Q1	Q2	Q3	Q4	Q5	Q6	Q7	TOTAL
		7	9	7	7	5	44.5
5	8	8	10	7	7	5	50

1. (5 marks) What makes a system call different from a signal handler? Tell me all about these differences in concise, but complete form.

system call:
- originates from a user application going to kernel
- code to execute lays in kernel space
- can contain many parameters

Signal handler:
- signal comes from kernel going to user process
- code to execute lays in userspace
- usually just a flag.

2. (8 marks) There are many different dimensions by which devices may be categorized. Choose 2 dimensions and describe the details of driver and application programming associated with each type of device along that dimension. For example: Dimension A: devices can be X or Y. For an X device type, the programmer must blah..... For a Y device type, the programmer must yaddah....

Dimension A: block vs. character

- for a block device, data is stored in blocks and can be typically be accessed in random order (eg. hard drive)
- For a character device, data comes in sequentially (linear ordering) and does not usually support going forward or backward (e.g. Keyboard, sound card)

Dimension B: polled vs. interrupt

For a polled device, we send a request to the device and have to wait for it to complete by continuously asking if it is done yet.

For interrupt driven, the device signals the kernel when the operation is complete.

3. (8 marks) Even for an individual device type, there are multiple access modes. Choose 2 device types and describe the access modes that are possible for that device type.

Hard Disk:

✓ ??

- direct access can be used to send commands to the drive (control signals)

- DMA can be used for the actual data transfer. We'd use this because it simplifies the transfer of large blocks of data

a little ~~not~~ off on organization and orthogonality of modes which is usually the way these things are described

7

~~Hd~~ Serial Port (UART):

- polled I/O for rapid "back and forth" communication (e.g. only a few characters at a time

- interrupt I/O for
  a) infrequent communication "let me know when something shows up"
  b) high-volume one-way traffic "one byte? come talk to me when you have 1000"

4. (10 marks) Write a system call that implements the IPC primitive Send(), with the followin semantics:

- send() messages are non-blocking. If the receiver is not waiting, then the message is added to a queue.
- each process can accept up to N messages. Note: this can be achieved by another system call SetMsgCapacity(), which you don't have to write. Assume this value is stored in a PCB variable of your choosing. If the receiver's queue is full, the sender must return immediately with an error code.

Use the following prototype for your system call, and include proper handling of race conditions, memory copying etc, in as close to C code as possible. The type SYSCALL is defined to be an enumerated type of 0 for success and negative integers for failure.

- assume there exists a task_struct "current"

~~- assume that, to save on calls to kmalloc, each process has a single allocated array of messages.~~ needs a length

- assume copy_from_user(src, dest)

```
SYSCALL sysSend (pid dest, void *sndMsg, int sndLen)
{
    task_struct* target;
    message* tosend;

    target = get_task_from_pid(dest);
    if (target == NULL)
        return -ENOSUCHPROCESS;
    P(target->msg_queue_lock);
    if(target->msg_queue_size <= target->msg_queue_current)
    {
        V(target->msg_queue_lock);
        return -ENOROOMONQUEUE;

    }
    tosend = ~~target~~ Kmalloc(sizeof(message));
    if(tosend == NULL)
    {
        V(target->msg_queue_lock);
        return -ENOMEMORY;
    }
    tosend->message = Kmalloc(sndLen)
    if(tosend->message == NULL)
    {
        V(target->msgqueue_lock);
        Kfree(tosend);
        return -ENOMEMORY
    }
    goto
    tosend->len = sndLen;
    tosend->sender = current;
    copy_from_user(sndMsg, tosend->message);
    queue_add(target->msgqueue, tosend);
    V(target->msg_queue_lock)
    return 0;
}
```

* could be improved if the queue were implemented as a ring buffer. then we'd only need one Kmalloc, for the message.

9

5. (7 marks) Describe and explain the operation of a boot loader and the startup sequence of a UNIX-based operating system. You can be quite general in the description, but focus on the boot process until *init* is active, including what must take place until a user may log on.

7

- assuming X86 32bit
- BIOS reads sector from boot device, jumps to it
- sector contains "stage 1", reads the kernel from the boot device, jumps to it
- Kernel sets up several data structures: global descriptor table, interrupt descriptor table, task-state segment, etc
- kernel hops into 32-bit protected mode
- kernel sets up internal data structures (e.g. task_structs) and minimal hardware/driver support (the rest is loaded from modules later on)
- the kernel creates the first process (init)
- do a "return" to user mode to begin execution of init

OK

6. (7 marks) Describe the detailed events that occur when a divid by zero exception is recognized by the CPU. Include all OS and application behaviours.

7

- CPU stores the %EIP of the instruction that caused the fault
- CPU flips to kernel mode and looks up the interrupt handler for Divide by zero handler, this push~~ing~~es status code onto stack and jumping to a general purpose exception handler
- general purpose exception handler sets up all context info about the current process and bounces to the real handler
- the real handler sends a signal to the process indicating the math error
- switch back to user mode
- application either traps the signal and deals with it, or dies due to the signal (default)

7. (5 marks) Priority inversion is a potentially deadly problem in an operating system. Describe 2 methods by which one can prevent it from occuring.

- promotion: if a low priority task (A) causes a high priority task (B) to block, temporarily promote A to the priority of B.

- Eliminate priorities altogether. Everyone is equal! Why should anyone get special privileges?! If you want something, wait your turn! *

5

——————— THE END———————

* Not a real serious answer, that seems like a bad idea ← ~~So, I guess~~ It does, but it would work